GPFS HPSS Integration: Implementation Experiences

Damian Hazen and Jason Hick
National Energy Research Scientific Computing Center at Lawrence Berkeley National Laboratory

Abstract

In 2005 NERSC and IBM Global Services Federal began work to develop an integrated HSM solution using the GPFS file system and the HPSS hierarchical storage system. It was foreseen that this solution would play a key role in data management at NERSC, and fill a market niche for IBM. As with many large and complex software projects, there were a number of unforeseen difficulties encountered during implementation. As the effort progressed, it became apparent that DMAPI alone could not be used to tie two distributed, high performance systems together without serious impact on performance. This document discusses the evolution of the development effort, from one which attempted to synchronize the GPFS and HPSS name spaces relying solely on GPFS's implementation of the DMAPI specification, to one with a more traditional HSM functionality that had no synchronized namespace in HPSS, and finally to an effort, still underway, which will provide traditional HSM functionality, but requires features from the GPFS Information Lifecycle Management (ILM) to fully achieve this goal in a way which is scalable and meets the needs of sites with aggressive performance requirements. The last approach makes concessions to portability by using file system features such as ILM and snapshotting in order to achieve a scalable design.

1 Introduction

The National Energy Research Scientific Computing (NERSC) center at the Lawrence Berkeley National Laboratory (LBNL) provides supercomputing resources to a large and diverse scientific community to aid in scientific discovery for the Office of Science within the Department of Energy (DOE). Often new computational systems at NERSC present new operating and storage system management challenges. While this provides for new computational capability, it also presents a challenge in terms of data management in that users are required to maintain different copies of their data on computational systems that do not interoperate. As datasets grow with the computational capabilities of compute clusters, maintaining multiple copies of data becomes increasingly costly both in terms of storage resources and in human effort required to effectively manage the data. In recent years, NERSC and other high performance computing centers have positioned themselves to take advantage of advancements in cluster file system technologies, and to deploy file systems that can be used across multiple compute platforms in the center [i,ii]. Goals of this approach include the "debalkanization" of storage resources, allowing for common procurement, deployment and use of what have traditionally been storage islands in data centers. Center wide file systems offer the potential of both scale and persistence at levels that surpass file systems tied to particular compute engines. Users see benefits in a center wide name space

mounted across major compute platforms, and in deduplication of their data. However long-term filesystem management including backup, recovery and enforcement of fair sharing of resources offers new challenges. For this reason, NERSC believes it is desirable to research and develop systems that integrate disparate file and storage systems to provide a common globally available logical view of a center's user data, but also to leverage storage technologies predicated on long term stewardship of data. One such effort at NERSC is the File and Archive System Integration project.

NERSC's effort to integrate a clustered parallel file system with an archival storage system originated around 1996 with joint interest from a small contingent of the High Performance Storage System (HPSS) collaboration. Participants included IBM, Lawrence Livermore and Lawrence Berkeley Laboratories. At this time, there were a series of meetings to discuss ideas for a design that would benefit the users at each site. This remained an informal and loosely organized research project where each site explored multiple different approaches from 1996 to 2005. With the deployment of the global file system at NERSC, and the successful production use of HPSS at the center, IBM and NERSC formalized the project by officially allocating development resources to work on the integration software. IBM's Global Parallel File System (GPFS) would be used as the file system front end, while HPSS would serve as the long-term archive.

The goals of the project can be stated simply:

- Provide a means to transparently migrate data between the GPFS file system and the HPSS storage system.

- After data migration, file system blocks allocated for the migrated file may be freed for reuse, allowing the file system to scale with the HPSS backing store.

- Upon access of migrated data, the data would be staged back into the file system, prior to returning control to the user's process.

- Provide for file system recovery using the data residing in HPSS.

2 Storage Technologies Used

Both GPFS and HPSS have customers in high performance computing, servicing some of the largest super computing centers in the world, and a number of centers use both offerings. This section provides a brief overview of the two systems.

2.1 HPSS Overview

HPSS is a data storage system that has been in production use since 1999 at a number of Department of Energy National Laboratories and other large computing installations. Its architecture is based on the IEEE Mass Storage System Reference Model [iii,iv,v], and implements a hierarchical storage management system, providing configurable, tiered data storage. The system manages file movement through the storage hierarchy of disk

and tape devices. Data hierarchies are constructed and migration and purge policies defined to reduce data storage cost, satisfy I/O intensive storage requirements, and to provide fault tolerance through redundancy. HPSS provides a hierarchical namespace giving users the ability to organize files in directories, use symbolic or hard links. It uses UNIX style permission settings to control access to objects in the namespace and allows for finer grained access control via access control lists (ACLs).

HPSS is used in many instances as a long-term archive and as such, metadata consistency, backup and recovery are paramount. Having centralized metadata with a reasonably small footprint, and a metadata engine, which provides the ability to take transactionally consistent backups of a live system, allows sites to safeguard their archive while maintaining system availability. HPSS uses the DB2 relational database for these features. HPSS consists of centralized services for metadata, space allocation and volume management and distributed data movers that can be scaled to satisfy I/O requirements in HPC environments. The system supports interfile and intrafile parallelism, handling many simultaneous I/O requests and striping I/O for individual files across devices. As file sizes and device capabilities increase, the system can be configured to stripe data across additional network, compute, disk and tape resources, allowing it to scale with the hardware infrastructure. HPSS supports data transfers over TCP/IP networks as well as 3rd party data movement in a SAN environment where the sources and sinks have direct access to the disk devices. The system supports a number of disk and tape technologies. Systems with 10's of petabytes of data and nearly 100 million files are currently in production [vi].

2.2 GPFS Overview

GPFS is a clustered parallel filesystem, which is used at some of the same large computational facilities as HPSS [vii]. Save for a few minor exceptions, it presents an image of a traditional POSIX compliant file system running on a stand-alone machine. A shared disk abstraction layer is used for shipping I/O requests to the disk servers. Data is striped across devices for high-speed access, and the client node uses read ahead/write behind techniques with a local cache for improving I/O rates. Metadata operations are journaled to the filesystem allowing journal recovery, in the event of node failure, to occur on another node in the cluster. Byte range locking is used to support intrafile parallel I/O operations. A token server is used to grant file access, space allocation and metadata rights allowing for distributed and parallel operation. A number of optimizations have been implemented reducing the number of token requests needed while still preserving POSIX semantics and file system consistency. Additionally, GPFS provides extended features that are key to the integration effort. These include an implementation of the XDSM DMAPI standard, file system snapshotting and Information Lifecycle Management (ILM) capabilities. These features are not universally provided by filesystems so each is described briefly below.

2.2.1 Data Management API (DMAPI)

The XDSM DMAPI specification gives file system implementers and writers of Data Management applications a set of file system extensions that may be used to implement an HSM system [viii]. Important features of the specification are:

- **DMAPI events.** Events provide notification of name space and I/O file system activities to the Data Management program.

- **Extended attributes.** Extended attributes are characteristics that can be associated with each file system object and contain information used by the Data Management Application. Common uses of extended attributes are for storing the object identifier for a file system object in secondary storage.

- **Managed regions**. Managed regions allow file data regions to be tagged such that access through standard POSIX I/O calls trigger event notification to the Data Management program.

Most XDSM implementations including the GPFS implementation consist of two components: a kernel extension through which filesystem activity passes, and a library that exposes the XDSM DMAPI to user space DM applications. DM applications receive notification of file system events via message queues, referred to as sessions, that are maintained by the XDSM kernel extension. Events can be either synchronous, which requires the application response or asynchronous. Synchronous events block the file system execution thread that generated the event until handled by the DM application. Delivery of asynchronous events is not guaranteed.

In order to better support DM applications in a parallel file system environment, GPFS has some extensions and semantic modifications to the XDSM DMAPI specification [ix]. Of particular interest are:

- **Clustered file system session support.** The XDSM specification makes an implicit assumption that the machine generating the event is the same machine that holds the session. With the GPFS DMAPI implementation, a session is assigned to a certain node in the cluster, and events generated throughout the cluster will be delivered to that session node. Events of differing types may be sent to independent sessions or session nodes.
- **Mount event handling**. GPFS delivers mount event notification for each mount and unmount operation on each node, and extends the mount event structure to indicate if the event is for the node running the DMAPI session, or for another node in the cluster. This allows the DM application to assert that the session node is the first to mount the file system and the last to unmount.
- **Failure recovery.** GPFS extends DMAPI session recovery to include session node failure. If a session node fails, the session can be recreated on another node and messages that were queued to the session can be recovered.

- **Parallel I/O.** The DMAPI I/O routines *dm_read_invis(), dm_write_invis(),* can run on any node in the cluster and can be used for parallel data movement within a file on file system block boundaries.

2.2.2 Information Lifecycle Management

Recent versions of GPFS incorporated ILM into the product. With ILM, storage devices can be partitioned into pools with certain traits. For example, a pool may be constructed out of devices that have particular reliability or performance characteristics [x]. To direct file placement and migration policy, GPFS uses an SQL like policy language. Policy can apply to initial file placement, indicating which pool the file exists in when first created, file migration, which allows data to be moved between pools en masse and file deletion. Placement policies are maintained in memory across all nodes in the cluster. Migration and deletion polices are described in a policy configuration file and are applied as a scheduled activity. To generate a list of candidates for file migration, GPFS ILM scans the file system building a result set of file attributes and pathnames that matches the search criteria specified. Traditionally, scanning large file systems is expensive. GPFS adapted file system metadata structures to better support high speed scanning. The scan, resulting sort, and merge workload is partitioned and distributed across all nodes in the cluster resulting in a fast, scalable scanning engine that makes scanning feasible in HPC environments. To illustrate, a recent internal IBM benchmark was able to scan one billion files in under 3 hours [xi].

2.2.3 Filesystem snapshots

Like some other file systems [xii,xiii], GPFS provides a copy on write file system snapshot feature. Taking a snapshot causes a static copy of the filesystem at the time of the snapshot to be maintained until it is administratively removed. Any changes to files or structures do not affect the snapshot. The snapshot can be scanned and ILM policies can be applied to it.

3 Previous experience with XDSM

The Distributed File System (DFS) backed by HPSS has been successfully deployed at several HPSS production sites including a large installation at Indiana University. Although the DFS product has been discontinued as an offering from IBM, it's instructive to review the system's architecture as an example of a successful DMAPI based solution. The software uses DMAPI implemented in DFS, but with some significant extensions [xiv]. The integrated system provided two modes of operation: one that consisted of traditional HSM functionality where data was migrated from the file system into HPSS. To access the data, the file would have to be read back onto the file system. The implementation also operated in a mode where the DFS name space was replicated in HPSS. In addition to allowing access to the backing store through the DFS file system, users could access migrated data through the HPSS client API. Accessing data directly through the HSM is not addressed in the XDSM specification and, guaranteeing archive and file system consistency for duel resident data introduced additional complexity.

To guarantee namespace synchronization, the DFS/HPSS implementation used a two-phase commit for namespace transactions. DMAPI namespace events provide event pairs for notifying a DM application of file system name space activity. An initial pre-event is delivered to the DM application indicating the intended operation, and a second post-event notification including status of the executed operation is sent once the action has been completed in the file system. DFS/HPSS operated by starting a transaction when the pre-event notification was received, and then either committing or rolling back the transaction depending on the status delivered in the second event. Further, DFS extended the DMAPI specification to provide event pairs for all event types including attribute events, adding an additional field in the pre and post event for pairing purposes, as well as providing guarantees of event delivery for the pre and post events except for the case of file system failure. In general, these features allowed the two namespaces to remain synchronized.

While this approach provides an effective means of synchronizing the file system and HPSS namespaces, it does introduce additional overhead associated with event delivery and transaction management. Additionally, because the DMAPI events generated by the file system are synchronous and block until responded to by the DM Application, file system object create rates proceed at roughly the combined rate of creates in the file system and creates in the archive. This is generally not an issue when creating large files, as the time it takes to perform I/O will dominate, however it is problematic for small files.

4 GPFS HPSS integration - DMAPI driven approach

Efforts to integrate GPFS and HPSS initially proceeded in a way similar to what was done for DFS. The project began by exploring designs using solely the DMAPI component in GPFS to implement the data migration and backup functionality. As with DFS, two approaches were developed: one which replicated the file system name space in HPSS, and another which pursued a more traditional HSM design, where the namespace would not be replicated. Instead, file system objects would reference HPSS objects by a special handle stored in a DMAPI extended attribute. In both cases, access to migrated files would only be via the file system. For the first approach, the name space replication would be used in providing a backup of the file system, as well as possibly providing read-only access through HPSS at some future date. For the traditional HSM design, a backup tool, which could interpret the application specific DMAPI attributes similar to what DMF, XFS and DFS provided would be developed. The tool would be aware of data that had been migrated to HPSS via HSM, so that backups could be taken without triggering a stage back to the filesystem.

A significant hurdle to overcome with either approach is the difference in namespace operation rates between GPFS and HPSS. HPSS is an archival data system, with emphasis placed on centrally managed metadata, long-term stability and recoverability. GPFS leverages the combined compute power of the clusters it runs on, allowing namespace operations to proceed without contacting a central service in many cases. This architectural difference results in a significant mismatch in namespace rates. Two

options were explored to attempt to address these differences. First, we examined if buffering events to a log, so that they could be worked off over time would be effective. Rather than performing the HPSS namespace operation in line as the events occurred in the filesystem, the software would coalesce and journal the event for later processing. Second, we examined minimizing the number and kind of namespace events we would set disposition and register for, so that only a small number of event types were generated. Exploration included registering only for the asynchronous 'post' namespace events, coalescing events of like types, and combining event registration with limited scanning techniques used to reconcile the two systems.

Eventually, we found these techniques to be insufficient for use in an HPC environment. While journaling could smooth over namespace bursts, it provides little help for file systems with high sustained namespace activity. Coalescing helped in some instances but introduced complexity. Registering for only asynchronous events introduced timing and event delivery failure problems. For these types of events, DMAPI makes a best effort to deliver them to the DM application's session, but makes no guarantees. Additionally, since most events originate off the session node and may be caused by parallel processes running across many nodes, events that are delivered out of order become a concern. In load testing, events would not always arrive on the DM application's session in the order they occurred, and the specification does not provide any non-opaque time stamp or sequence data that could be used to reconstruct the event sequence.

Overriding complexity concerns were concerns about the DMAPI architecture on GPFS. Recall that for a file system in a GPFS cluster, all events of a particular type must be delivered to the same session. A DMAPI session resides on a single node and provides a queue maintained in kernel memory for holding file system events until they can be consumed by the DM application. A single queue on one node in the cluster handling events from many generators presents a fan-in problem. During load testing, as the number of events generated by the cluster exceeded the session node's ability to process them, kernel memory was consumed until the file system was forced down on the session node.

The tight coupling that DMAPI imposes between the file system and tertiary storage, particularly with namespace events, results in significant communication between session and client nodes. GPFS use of a single session node per file system for an event type presents load concerns on the session node, as well as an event backlog on the client nodes delivering the messages to the session if the events are not handled quickly. Without special provision, DMAPI requires that the system managing tertiary storage be available in order for the filesystem to continue to operate. This imposes operational difficulties on a data center. In the end, we concluded that while there were certain application domains where DMAPI provides a useful set of primitives for constructing HSM functionality, there were serious limitations both in terms of the tight coupling between file system and archive that the event model introduces, and its lack of support for high performance clustered file systems. These along with other problems in the specification discussed in [xiv] and [xv] make writing well performing, low overhead DM applications difficult. It was time to examine another approach.

5 Hybrid Approach – DMAPI and ILM

In 2006 the project abandoned attempts to build a standards based solution using the XDSM specification. Instead, in collaboration with IBM's Almaden Research Center we began development of an approach that minimizes DMAPI use, and relies instead on ILM and snapshotting for some key functional components. While this change represents a capitulation in terms of portability, it addresses much of the concerns in performance and scalability that an HPC workload requires.

Recall that the GPFS ILM allows administrators to configure pools of storage with certain characteristics, and to migrate files between storage pools. Initially, storage pools were solely managed by GPFS, however, the idea of an external pool, managed by archival software such as HPSS is a key piece in the ILM driven design. In this case, HPSS takes on the role of manager for an external storage pool. Candidate lists for migration are delivered to HPSS, which uses its high performance I/O capabilities to move data into tertiary storage. This differs from the first approach, where DMAPI events were used to drive the system, and were required for maintaining a migration candidate list. With the ILM driven approach, the ILM policy engine together with file system snapshotting is used to discover candidates for migration, removing the need to track file system namespace operations. DMAPI is only used for trapping user I/O requests that require access to HPSS.

In the initial approach, backup was a relatively minor component, and consisted primarily of capturing the file system namespace, and relying on the copy of the data that resided in tertiary storage. It was missing key features needed in a robust backup solution, including file versioning and point in time backup consistency. In the ILM driven approach, backup is expanded to include these features, relying on ILM and snapshotting to build a richer featured backup solution. The next two sections describe the HSM and backup operations of the ILM driven approach.

5.1 HSM

HSM uses GPFS ILM to generate a migration candidate list. To produce the list, the policy manager runs periodically with a set of rules that describe the attributes of candidates to be migrated from GPFS managed storage pools into an external HSM pool managed by HPSS. To pick migration candidates, most of the traditional fields maintained in a file's inode, as well as file pathnames with pattern matching filters can be used as predicates to the selection clause. From these rules, the policy manager generates a list of files that are candidates for migration into the HPSS managed pool, and passes the list to a set of HPSS client programs that are used to perform the data movement. Data can be pre-migrated with copies existing in both HPSS and GPFS, or migrated where the file's inode, extended attributes and a small amount of data is left in GPFS, but all other file system blocks are freed for reuse. The file system name space is not replicated in HPSS as part of HSM. DMAPI is used in a limited fashion, primarily to deliver synchronous notification to the system when user activity in the file system

requires access to data stored in HPSS so that data can be moved back into the file system, and for tracking file system mount and unmount events.

While GPFS ILM provides a high performing approach for generating a candidate list for migration that does not have the scaling or fan-in problems present with DMAPI, there still exists the disparity in metadata rates between HPSS and GPFS. To deal with this, the ILM driven approach introduced aggregation in HPSS where small files are combined into larger storage units in HPSS. For aggregation, the project chose to use HTAR – a standard based HPSS client application that is currently in use at a number of HPSS sites [xvi]. HTAR, groups files into configurable sized aggregates written in USTAR format. The tool uses HPSS parallel I/O for fast data movement. HTAR provides a separate aggregate index that can be used for efficient lookup and access of individual aggregate members. The combined system, using GPFS ILM for generating HSM migration candidates and HTAR for aggregation was demonstrated at the Super Computing Conference in Reno, Nevada in November 2007. The demonstration showed that a billion 0 byte files in GPFS could be migrated to HPSS in less than 24 hours using the GPFS ILM to generate the candidate list, and the aggregation feature that HTAR provides [xvii].

5.2 Backup

Since HSM does not save the file system namespace or attributes, a separate backup feature is provided to allow for recovery in the event of file system failure. Backup consists of everything needed to restore the file system from the ground up - file system namespace, extended attributes as well as GPFS cluster metadata is captured. For discovering backup candidates and migrating into HPSS, the system again uses the ILM policy engine and the logical construct of an external storage pool managed by HPSS. A separate set of policy rules is used to generate backup candidates. To guarantee a consistent backup image, backups are taken from a GPFS read only file system snapshot. The snapshot can be used to determine the list of files that have changed since the previous backup run. Backup and HSM use the same file data in HPSS, so under normal operation, most of the I/O required to take a full backup of the file system will have already been completed by HSM. For a backup, all data blocks for any GPFS candidate file that does not have a current copy in HPSS is flushed to tertiary storage. The inode content, ACLs and other extended attributes for all objects in the file system (files, directories, symbolic links, etc.), and file system configuration (cluster, disk, etc.) is stored. Since data is read from a static snapshot, the backup operation is idempotent and can be repeated until an internally consistent point in time image of the file system has been captured.

Since backup and HSM use the same HPSS data objects, this introduces the need to store file versions. For example, if a file is HSM'd into HPSS and is also part of a backup image, and that file is modified in the file system, a new version of the file will need to be created in HPSS to preserve the validity of the backup. To address this, GPFS provides a version identifier - called an epoch number - as part of their inode metadata. It is a monotonically increasing value that is associated with a snapshot. After a snapshot of the file system is taken, any creation or modification of a file system object will result in the

epoch number for that object being assigned a value of the previous snapshot number plus one. The combination of inode, inode generation number and epoch determines a unique file version.

6 Additional Concerns

While ILM provides a means for managing migration and backup candidate lists, there are still several challenges that need to be met. This section briefly describes areas requiring additional work.

6.1 File System Deletions

An HPSS object cannot be deleted while there are still references to it. This includes both file system references and references from backup images. To address this, cleaning up unreferenced HPSS objects is designed to be a separate activity using a garbage collection scheme. Files in HPSS will only be deleted by garbage collection when no backup or file system references exist. Since a deleted file may be referred to by any number of backup images, depending on backup retention policy, an efficient means of determining the set of backup images a file is a member of is required. Aggregates complicate the issue as well, since an aggregate cannot be deleted until no references exist to any of its members. During design, we explored a number of metadata layouts that aimed to track references in backups to objects in tertiary storage, keeping in mind the need for both time and space efficiencies.

6.2 Metadata Residence

For any approach, metadata that provides mapping from file system entries to counterparts in tertiary storage is required. A common method of doing so, and one that the DMAPI specification supports is to store a handle to the object as a file's extended attribute. But, since a file system reference is only one of possibly many references that can exist to an object in HPSS, the reference may need to be maintained after the file system object has been deleted. One approach to address this is to maintain the mapping in the file's extended attribute, and when the file is deleted, to log the delete so that a background process can search for references to the object in backup images as part of garbage collection. A downside to this approach is that the DM Application requires deletion event notification. Alternatively, we considered storing all mapping metadata in an HPSS allocation table. With this approach, results from subsequent file system scans would be compared to determine potential deletion candidates. Before actually removing the object in HPSS, the backup records would also need to be consulted.

6.3 File System Restoration

As discussed, the design uses the same backing store for HSM as for backup. If a file system needs to be restored, first the GPFS cluster metadata is reconstructed, and then the namespace is recovered. While all data will be accessible at this point, the design calls

for tracking whether the restored file was resident on the file system, and if so then it is copied back to GPFS as part of the restore process. Performing a bulk repopulation during restore rather than relying solely on user access to populate the file system will let the system order requests so that tape accesses are handled more efficiently. To further improve efficiency, the design calls for being able to repack tapes in order to minimize the number of tapes a full file system restore requires.

6.4 Partial File Migration

For extremely large files, there are efficiencies that the project did not examine in detail, but have the potential for significantly improving performance of the system. One way of doing this is using multiple managed regions per file. A GPFS file can have up to 32 DMAPI managed regions where each managed region corresponds to a contiguous range of bytes within a file which can be set to generate synchronous read, write and truncate events whenever one of those operations are performed within the managed region. Using multiple managed regions can be beneficial for large files in that it lets the system stage only part of a file back to the file system when accessed, and gives a method for resuming migration of a large file to HPSS without having to re-migrate the entire file if the migration failed before completion.

Conclusion

The DMAPI specification provides a set of primitives that can be used to construct an HSM system. There have been a number of HSM implementations based on DMAPI [xviii,xix,xx] including software that used DMAPI to link DFS and HPSS. While DFS is a distributed file system, it is not a clustered file system, and a file's metadata belong to a single server. GPFS is a clustered file system that strives to allow distributed, parallel operations for both metadata and I/O. The GPFS DMAPI implementation provides extensions to address some of the needs of clustered file systems such as multiple sessions, parallel invisible reads and writes, and cluster mount support, however it makes concessions to the DMAPI specification and to existing DM applications. For example, rather than having events delivered and handled on the node that generated the event, requiring a distributed DM application, it defines a single session for handling events of like type. This architecture presents scaling issue for HPC workloads.

We found that GPFS ILM provides an elegant solution to the event-scaling problem. Rather than rely on namespace events to maintain a migration candidate list, the system uses ILM to deliver candidates to the DM application. The candidate discovery is performed in parallel across all nodes in the GPFS system, delivering a few large lists of migration candidates. This eliminates DMAPI event traffic for everything but I/O where access to tertiary storage is needed. Aside from I/O events, DMAPI constructs such as managed regions, extended attributes and exclusive/shared access rights are still useful in the system.

As an archival storage system, HPSS metadata rates are significantly slower than GPFS. The design calls for using the HTAR aggregation tool to perform HPSS side aggregation to reduce the number of HPSS objects that must be created. HTAR allows the HPSS

system to keep pace with the file system's create rates, and still perform striped data transfers to the archive.

Large, platform independent clustered file systems are relatively new introductions to the HPC center. As of yet, the file systems do not have integrated backup solutions that scale. Additionally, backup solutions that are used in traditional DMAPI implementations such as DFS/HPSS, DMF and CXFS generally provide a tool that is aware of the DM Application specific attributes that indicate a file has been moved to tertiary storage, and capture the namespace for restoring the file system in case of failure. As the number of devices and concerns of undetected data corruption increase, we believe that there's value in providing a robust backup solution. The solution uses the GPFS file system snapshot for guaranteeing point in time consistency. Multiple copies of a file can exist in the HSM, depending on a site's retention policy.

Acknowledgment

This work was produced by the University of California, Lawrence Berkeley National Laboratory (LBNL) under Contract No. DE-AC02-05CH11231 with DOE.

---

[i] W. T. C. Kramer, A. Shoshani, D. A. Agarwal, B. R. Draney, G. Jin, G. F. Butler, J. A. Hules. *Deep scientific computing requires deep data.* IBM Journal of Research and Development 48(2):209- 232, March 2004.

[ii] Greg Butler, Rei Lee, Mike Welcome. *GUPFS: The Global Unified Parallel File System Project at NERSC.* Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies, April 2004, pages 361-371.

[iii] *IEEE Mass Storage System Reference Model V5 (MSSRM).* http://www.ssswg.org/public_documents.html.

[iv] Richard W. Watson. *High Performance Storage System Scalability: Architecture, Implementation and Experienc*e. Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies, April 11-14, 2005, pages 145-159.

[v] Danny Teaff, Dick Watson, Bob Coyne. *The Architecture of the High Performance Storage System (HPSS).* Proceedings of the Goddard Conference on Mass Storage Systems and Technologies, March 28-30, 1995, pages 45-74.

[vi] *HPSS Petabyte Club.* http://www.hpss-collaboration.org/hpss/about/PBdecimal20080317.pdf

[vii] Frank Schmuck, Roger Haskin. *GPFS: A Shared-Disk File System for Large Computing Clusters*. Proceedings of the Conference on File and Storage Technologies, January 28-30, 2002, Monterey, CA, pages 231-244.

[8] *Systems Management: Data Storage Management (XDSM) API.* Technical Standard. Open Group CAE Specification, Open Group, February 1997.

[ix] *GPFS V3.1 Data Management API Guide.* First Edition (April 2006), IBM, 2455 South Road, Poughkeepsie, NY 12601.

[x] *GPFS V3.1 Advanced Administration Guide.* First Edition (April 2006), IBM, 2455 South Road, Poughkeepsie, NY 12601.

[xi] *IBM Supercharges Management of Massive Amounts of Data -- A Billion Files at Lightning Speed*. IBM Press Release, Armonk N.Y., October 02, 2007. http://www-03.ibm.com/press/us/en/pressrelease/22405.wss

[xii] Dave Hitz, James Lau, Michael Malcom. *File System Design for an NFS File Server Appliance.* Proceedings of the USENIX Technical Conference. 2004, San Francisco, CA, pages 235-246.

[xiii] *Solaris ZFS Administration Guide.* Part Number 817-2271, Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, CA 95054, June 2008.

[xiv] Rajesh Agarwalla, Madhu Chetuparambil, Craig Everhart, T.N. Niranjan, Rena Haynes, Hilary Jones, Donna Mecozzi, Bart Parliman, Jean E. Pehkonen, Richard Reuf, Benny Wilbanks, Vicky White. *HPSS/DFS: Integration of a Distributed File System with a Mass Storage System.* Proceedings of the Sixth Goddard Conference on Mass Storage Systems and Technologies, March 1998, pages 57-70.

[xv] Miroshnichenko, Alex (1996). *Data Management API: the standard and implementation experiences.* AUUG 96 & Asia Pacific World Wide Web 2nd Joint Conference Proceedings, Sep. 18-20, 1996. http://www.csu.edu.au/special/auugwww96/proceedings/alex/alex.html

[xvi] HTAR. http://www.mgleicher.us/index.html/htar/

[xvii] *HPSS for GPFS at SC07.* http://www4.clearlake.ibm.com/hpss/about/SC07.jsp

[xviii] *DMF Release and Installation Guide.* Silicon Graphics, Inc., 1600 Amphitheatre Pkwy., Mountain View, CA 94043.

[xix] *VERITAS NetBackup Storage Migrator for UNIX v4.5.* Whitepaper, 2002. VERITAS Software Corporation, Corporate Headquarters 350 Ellis Street, Mountain View, CA 94043.

[xx] *IBM Tivoli Storage Manager for Space Management for Unix User's Guide, Version 5 Release 2.* IBM Corporation, 11400 Burnet Road, Austin, TX 78758.